Katedra informatiky
Přírodovědecká fakulta
Univerzita Palackého v Olomouci

BAKALÁŘSKÁ PRÁCE

2D RPG hra v Unity Engine

2025

Jan Jaroš

Vedoucí práce:
Mgr. Eliška Foltasová

Studijní program: Informatika,
Specializace: Programování a vývoj software

Bibliografické údaje

Autor:	Jan Jaroš
Název práce:	2D RPG hra v Unity Engine
Typ práce:	bakalářská práce
Pracoviště:	Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci
Rok obhajoby:	2025
Studijní program:	Informatika, Specializace: Programování a vývoj software
Vedoucí práce:	Mgr. Eliška Foltasová
Počet stran:	37
Přílohy:	elektronická data v úložišti katedry informatiky
Jazyk práce:	český

Bibliographic info

Author:	Jan Jaroš
Title:	2D RPG game in Unity Engine
Thesis type:	bachelor thesis
Department:	Department of Computer Science, Faculty of Science, Palacký University Olomouc
Year of defense:	2025
Study program:	Computer Science, Specialization: Programming and Software Development
Supervisor:	Mgr. Eliška Foltasová
Page count:	37
Supplements:	electronic data in the storage of department of computer science
Thesis language:	Czech

Anotace

Tato práce se zabývá vývojem 2D hry žánru RPG. Popisuje základní prvky prostředí Unity Engine a dále se věnuje procesu vývoje samotné hry, včetně realizace jednotlivých herních mechanik. V závěrečné části je představena výsledná hra včetně všech implementovaných funkcionalit.

Synopsis

This bachelor's thesis focuses on the development of a 2D RPG game. It describes the fundamental elements of the Unity Engine environment and then delves into the game's development process, including the realization of individual game mechanics. In the concluding section, the finished game is presented along with all implemented functionalities.

Klíčová slova: Unity Engine; C#; 2D videohra; RPG

Keywords: Unity Engine; C#; 2D video game; RPG

Rád bych poděkoval své vedoucí práce Mgr. Elišce Foltasové za pomoc a cenné rady, které mi poskytla během realizace této bakalářské práce.

Odevzdáním tohoto textu jeho autor/ka místopřísežně prohlašuje, že celou práci včetně příloh vypracoval/a samostatně a za použití pouze zdrojů citovaných v textu práce a uvedených v seznamu literatury.

Obsah

Úvod

Cílem této bakalářské práce bylo navrhnout a implementovat 2D RPG hru v Unity Engine. Zkratka RPG vychází z anglického výrazu *role-playing game*, což lze do češtiny přeložit jako „hra na hrdiny“. RPG hry představují herní žánr, ve kterém hráč přebírá roli herní postavy, která ho provází herním světem. Typickými rysy RPG her jsou získávání zkušeností, vylepšování schopností a správa inventáře.

Pro vývoj této hry byl zvolen nástroj Unity Engine, a to zejména díky jeho rozsáhlé podpoře pro vývoj 2D her, možnostem skriptování v jazyce C#, dostupnosti široké knihovny assetů a aktivní komunitě vývojářů. V rámci vývoje byly implementovány základní RPG prvky, jako jsou systém předmětů a inventáře, zkušenostní systém, dovednosti či nepřátelské entity.

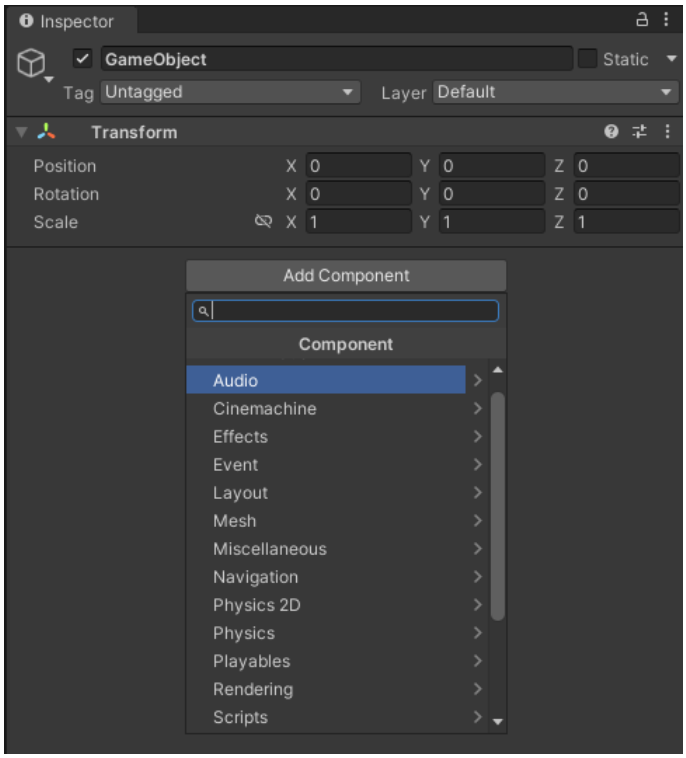
1 Seznámení s Unity Engine

Unity Engine [1] (dále jen *Unity*) je multiplatformní herní engine vyvinutý společností Unity Technologies, který slouží k tvorbě 2D a 3D her, simulací či interaktivních aplikací. Tyto softwary mohou být vyvinuty pro různé platformy – PC, konzole, mobilní zařízení, virtuální realita nebo webové aplikace.

Unity nabízí kompletní sadu nástrojů pro tvorbu grafiky, animací či fyziky herních objektů. Umožňuje rychlý a flexibilní vývoj díky vestavěným nástrojům a možnosti opakovanému použití objektů.

1.1 Komponenty

Komponenty [2] jsou funkční částí každého objektu, které určují jeho vlastnosti a chování. Každý herní objekt musí mít základní komponentu *Transform*, která určuje pozici objektu v herní scéně. Objektům lze přidávat další komponenty (v inspektoru tlačítkem Add Component, viz obr. 1).



Obrázek 1: Přidání komponenty

1.2 Assety

Všechny položky, které jsou součástí projektu a využívají se při tvorbě hry, se označují jako *assety* [3]. Jedná se o datové soubory, ze kterých se ve hře skládají jednotlivé prvky, jako grafika, zvuky nebo skripty. Assety lze vytvářet přímo

v prostředí Unity, importovat z externích zdrojů (např. z grafických editorů) nebo stahovat z obchodu *Unity Asset Store*, který poskytuje velké množství balíčků. Tyto assety jsou následně organizovány v rámci projektu ve složce `Assets`, odkud je lze dále využívat při vývoji hry.

1.3 Skripty

Skripty [4] jsou klíčovým nástrojem pro definování logiky a funkcionality ve hře. Pokud chceme objektu přidat funkcionalitu danou skriptem, připojíme daný skript hernímu objektu jako komponentu. Tím získá vlastnosti definované skriptem. Každý skript reprezentuje novou třídu, u které pokud chceme, aby nabýval funkcionalit herního enginu, dědíme ze třídy `Monobehaviour`.

1.3.1 Třída Monobehaviour

Třída `Monobehaviour` umožňuje herním objektům dynamicky komunikovat s herním světem, reagovat na změny či vstup uživatele. Mezi základní použití patří například metoda `GetComponent`, pomocí které můžeme získat odkaz na komponentu, kterou má herní objekt a můžeme tak za chodu programu měnit vlastnosti a funkcionalitu této komponenty.

1.3.2 Funkce událostí

Funkce událostí představují sadu vestavěných funkcí, na které mohou skripty dědící z třídy `Monobehaviour` reagovat implementací příslušných metod, označovaných jako zpětná volání. Jakmile dojde k určité události, Unity vyvolá odpovídající zpětné volání, na které lze ve skriptu zareagovat. Mezi tyto zpětná volání patří například uživatelský vstup nebo fyzika herních objektů.

1.3.3 Životní cyklus herních objektů

Životní cyklus herního objetu [5] je znázorněn na diagramu na obr. 2. Mezi nejčastěji využívané metody třídy Monobehaviour patří:

- `Awake` – První volaná metoda, která se spouští ihned po načtení instance skriptu (tedy před samotným spuštěním hry). Typicky slouží k inicializaci proměnných a nastavení výchozích stavů objektu.
- `Start` – Tato metoda je volána před prvním snímkem metody Update. Obvykle se používá pro inicializaci komponent nebo načtení dat.
- `Update` – Metoda volána při každém snímku. Slouží pro logiku, která se má vykonávat neustále – například pohyb objektu, vstup uživatele, kontrola herních podmínek nebo správa časovačů.

- `OnDestroy` – Metoda spuštěná při zničení objektu. Využívá se pro uložení dat, uvolnění zdrojů, případně k dalším operacím souvisejících s ukončením cyklu herního objektu.

1.4 Předloha objektu

Důležitým konceptem Unity je předloha objektu (nazývaná *Prefab* [6]), která umožňuje sestavený herní objekt uložit jako šablonu a poté opakovaně využívat – přetažením do scény nebo dynamickou inicializací pomocí skriptu.

1.5 Skriptovací objekty

Skriptovací objekty [7] představují speciální typ tříd v Unity, které slouží jako kontejnery pro data. Pro jejich použití je nutné vytvořit třídu, která dědí ze základní třídy `ScriptableObject`. Na rozdíl od běžných skriptů se skriptovací objekty nepřipojují jako komponenta k hernímu objektu,ale vytvářejí se jako samostatné assety, které se ukládají přímo do assetu projektu. Hlavní předností skriptovacích objektů je možnost uchovávat sdílená data na jednom místě, na které se odkazují různé části hry. Díky tomu není nutné opakovaně nastavovat stejné hodnoty u jednotlivých herních objektů.

1.6 Serializace

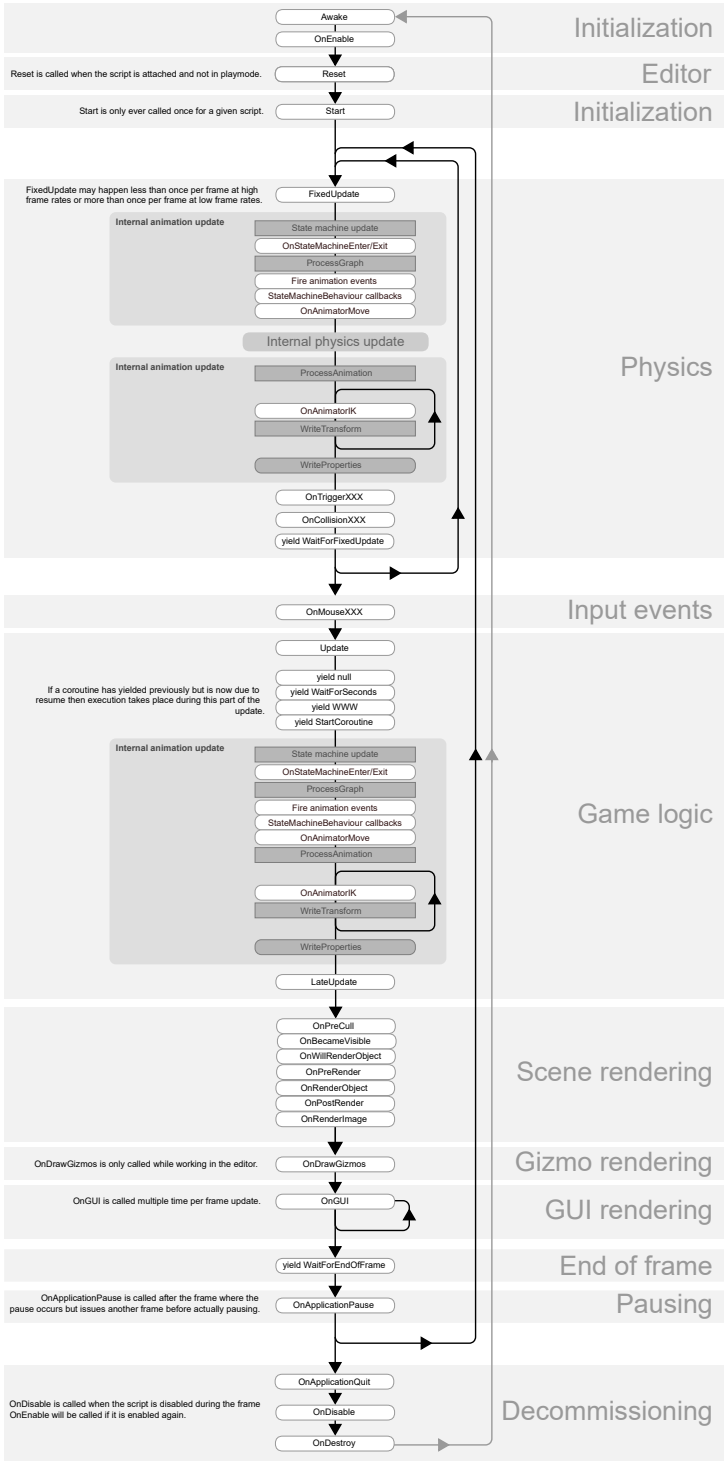
Serializace [8] v Unity představuje důležitý proces při vývoji her, během něhož jsou datové struktury a stav objektů převedeny do formátu, který je možné uložit a později znovu načíst. Unity tento proces využívá například při ukládání a načítání scén, assetů nebo předloh objektů. Pro vývojáře je však nejvíce přínosný při zobrazování a úpravě hodnot v inspektoru editoru, kde umožňuje manipulaci s daty přímo z uživatelského rozhraní editoru.

1.6.1 SerializeField

Ve výchozím nastavení Unity serializuje všechna veřejná pole ve třídách, které dědí ze třídy `Monobehaviour` nebo `ScriptableObject`. Pokud ale potřebujeme, aby bylo soukromé pole viditelné a editovatelné v inspektoru, použijeme atribut `[SerializeField]`. Tímto způsobem zůstává zachováno zapouzdření – pole je nadále přístupné pouze v rámci třídy, ale zároveň je možné hodnotu nastavovat přímo v editoru (kde se tato hodnota uloží spolu se scénou nebo předlohou objektu).

1.7 Koprogram

Unity disponuje speciálními typy metod – koprogramy [9] (z anglického názvu *Coroutine)* s návratovým typem `IEnumerator`. Tyto metody dokáží pomocí



Obrázek 2: Vývojový diagram životního cyklu herních objektů

klíčových slov `yield return` dočasně pozastavit svůj běh. A to například pomocí třídy `WaitForSeconds(n)` nebo `WaitForSecondsRealtime(n)`, kde n představuje dobu čekání ve vteřinách. V prvním případě se jedná o herní čas (ten může být pozastaven například v inventáři), ve druhém běží časovač v reálném čase. Koprogramy se spustí metodou `StartCoroutine`.

1.8 Scény

Základní jednotka, která obsahuje prostředí hry, se nazývá scéna [10]. Scény rozdělují projekt na logické části, což usnadňuje správu a organizaci obsahu. Typickými příklady jsou jednotlivé herní úrovně, obrazovky hlavního menu nebo nabídky nastavení.

2 Proces vývoje

2.1 Animace

Animace jsou nedílnou součásti her, neboť umožňují ze *spritů* („sad“ dvojrozměrných obrázků) vytvořit plynulý pohyb. Pro práci s animacemi musí mít herní objekty komponentu animátor. Tato komponenta obsahuje *Animator Controller*, který slouží jako „správce“ animací herního objektu. Ve hře má každá entita, jak samotný hráč, tak i každý typ nepřítele svůj Animator Controller. Animator Controller [11] u hráče se skládá z osmi hlavních podstavů (viz obr. 3), kde každý podstav je rozdělen na tři větve (viz podstav nečinného stavu na obr. 4). Tyto větve určují animace na základě zbraně, kterou je hráč vybaven. U nečinného stavu a stavu běhu jsou navíc tyto větve rozšířené o stavy míření a střelby. Tyto animace jsou postaveny na principu konečného automatu. Každá animace reprezentuje jednotlivý stav.

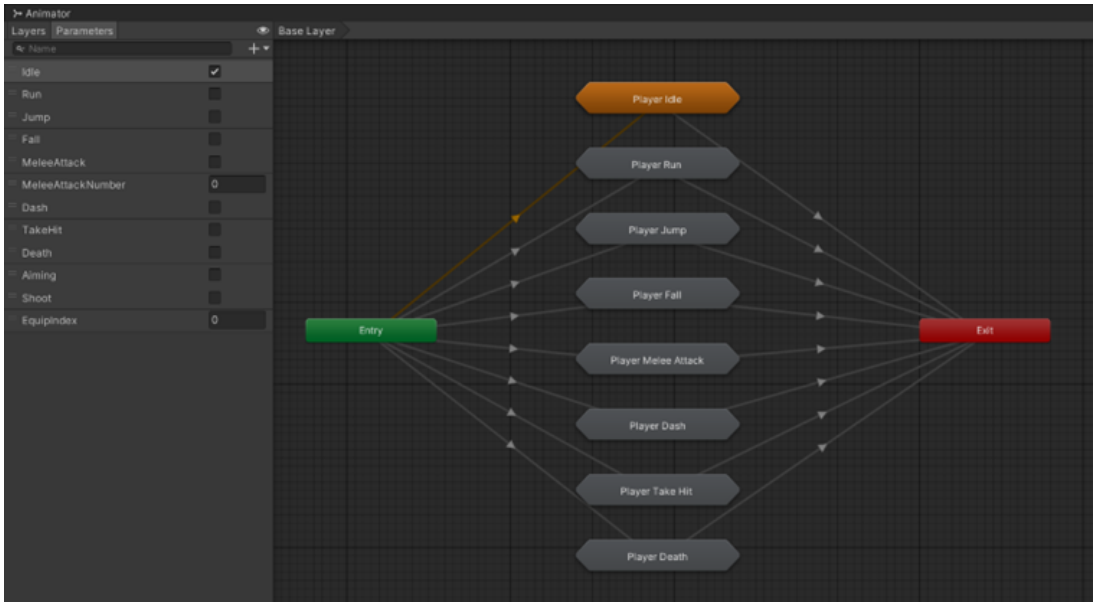
2.1.1 Řídící parametry

Tím, že jsou animace postaveny na principu konečného automatu, musí být zajištěny přechody mezi jednotlivými stavy. Tyto přechody jsou řízeny prostřednictvím parametrů (patrné v levé části obr. 3). Parametry jsou proměnné buď typu bool, který představuje druh stavu (např. nečinný, běžící, střílecí stav), nebo celočíselný typ, jenž představuje buď index vybavené zbraně, nebo slouží jako počítadlo představující útočné kombo.

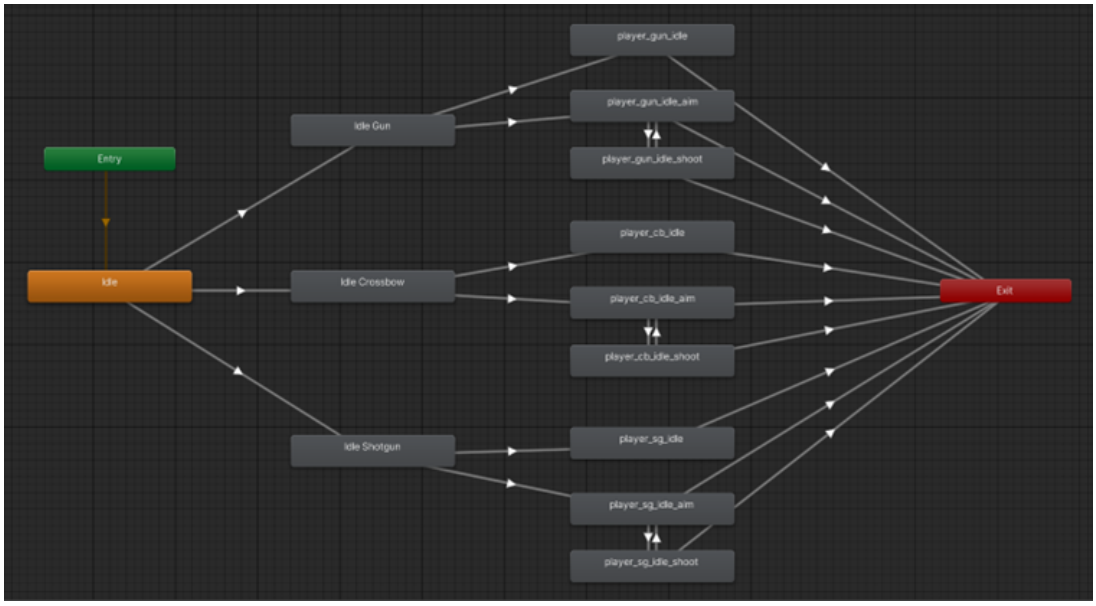
2.2 Entity

Veškeré postavy ve hře (hráč i nepřátelé) dědí ze skriptu `Entity`. Tento skript udržuje dvě komponenty:

- `RigidBody2D` [12]– Základní komponenta pro fyzikální vlastnosti, která zajišťuje působení gravitační síly a kolize.



Obrázek 3: Animátor



Obrázek 4: Nečinný animační podstav

- `Animator` [13]– Komponenta spravující animace, která slouží například pro nastavení hodnot řídících proměnných a následné přepnutí animací.

Dále tento skript spravuje pohyb prostřednictvím nastavení rychlosti a metody `FacingController`, která kontroluje orientaci entity během pohybu.

2.2.1 Player

`Player` je základní skript hráče zodpovědný za veškerou herní logiku, včetně inicializace konečného automatu animací a všech jeho stavů. U každého jednotlivého stavu nastavuje jméno řídící proměnné animátoru. Další významnou části je střílení, které realizuje metoda `ShotObject`. Ta nejprve zjistí, jakou zbraní hráč vystřelil, následně zjistí, jaké poškození uděluje a minimálně sníží peněžní hodnotu zbraně. Poté zavolá metodu, která odpovídá typu vystřelené zbraně: pro střelu z pistole metodu `ShotBullet`, pro střelu z kuše metodu `ShotArrow` a pro výstřel z brokovnice metodu `ShotCartridge`. Tyto tři metody nastaví výchozí pozici střely a následně vytvoří instanci střelného objektu z předlohy objektu střely. Poté přidá funkcionalitu komponentou `GunBullet`, fyzikální vlastnosti s rychlostí výstřelu a na závěr přehraje audio výstřelu. Ukázka příkladu zjednodušeného skriptu výstřelu z pistole:

```
1  private void ShotBullet1(float weaponDamage)
2  {
3      // Ensure correct facing
4      Quaternion rotation = Quaternion.Euler(
5          0,
6          0,
7          (actualDirection > 0) ? 0f : 180f);
8
9      // Create new instance of bullet
10     GameObject bulletGO = Instantiate(
11         bulletPrefab,                    // Gameobject to
               instantiate
12         bulletStartPosition.position,    // Start position
13         rotation,                        // Right rotation
14         bulletsParent.transform);        // Set parent to organize
               bullets in hierarchy
15
16     // Initialize the bullet's logic
17     GunBullet bullet = bulletGO.GetComponent<GunBullet>();
18
19     // Initialize bullet with player reference and damage value
20     bullet.Initialize(this, weaponDamage);
21
22     // Give the bullet its velocity so it flies in the chosen
           direction
23     Rigidbody2D bullet_rb = bulletGO.GetComponent<Rigidbody2D>();
24     bullet_rb.velocity = new Vector2((int)actualDirection, 0) *
           bulletSpeed;
25
26     // Play shot audio after fire
27     AudioManager.instance.PlayGunShotSound();
28 }
```

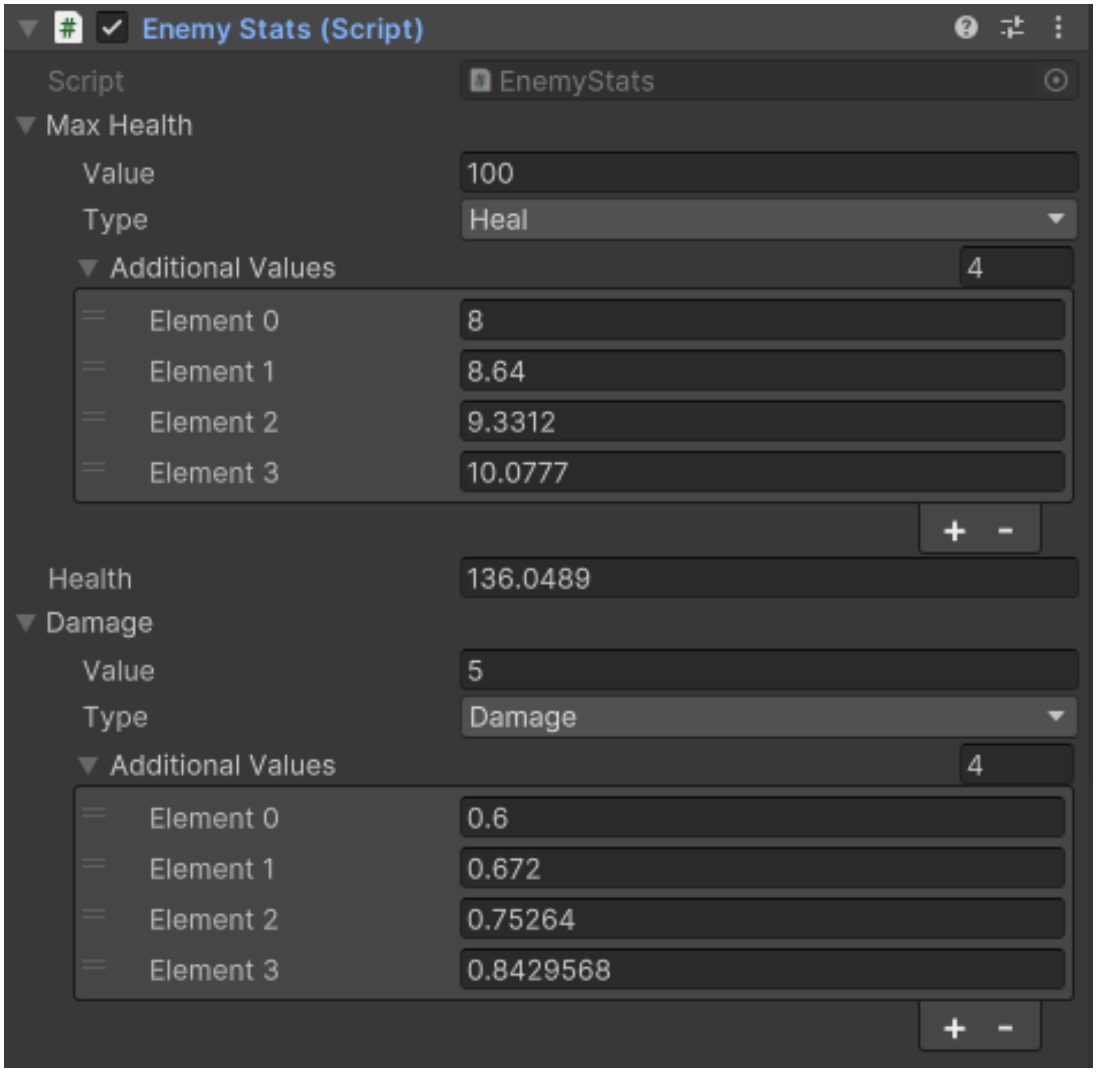
Zdrojový kód 1: Implementace metody pro střelbu

2.2.2 Enemy

Ze skriptu `Enemy` dědí veškeré nepřátelské jednotky. Disponuje stejnou funkcionalitou jako skript `Player`, přičemž navíc implementuje interakci s hráčem prostřednictvím proměnné `distanceToAttackPlayer`, která určuje maximální vzdálenost, v níž se hráč musí nacházet, aby na něj nepřítel zaútočil. Skript navíc obsahuje metodu `Die`, která zajistí po eliminaci nepřítele vypadnutí předmětů, spočítá kolik zkušeností hráč získá a ve finále nastaví nepřítele jako neaktivního. Objekt se ovšem nezničí, protože by se smazaly i jeho statistiky. Zničení objektu nepřítele by při oživení hráče způsobilo nekompatibilitu u nastaveni zpětných statistik. Při oživení nepřítele se nastaví jeho místo zrození na původní místo, obnoví se jeho zdraví, nastaví se jako aktivní a přejde do nečinného stavu.

2.3 Statistiky

Jednotlivé statistiky reprezentuje skript `Stats`. Ten má vnitřní hodnotu proměnné `value`, která uchovává počáteční hodnotu. Pro zvýšení celkové hodnoty statistiky slouží metoda `AddValue`, která přidá dodatečnou hodnotu do seznam a metoda `TotalValue` vrátí celkovou hodnotu statistiky. Ve hře je celkově šest typů statistik - none, cena, poškození, rychlost útoku, zdraví a zkušenosti. Na obr. 5 je znázorněno nastavení statistik nepřítele, kde figurují dvě statistiky – maximální zdraví a poškození. Obrázek obsahuje počáteční hodnotu `value` a přidané hodnoty v seznamu dodatečných hodnot.



Obrázek 5: Statistiky nepřítele

2.4 Statistiky entit

Statistiky hráče i nepřátel vycházejí ze třídy `EntityStats`. Tato třída obsahuje hodnotu zdraví a dále obsahuje statistiku maximálního zdraví. Nepřátelé mají

rozšířené statistiky pouze o základní poškození, kdežto hráč o poškození ze všech typů zbraní.

Událost `public event Action OnMaxHealthChanged`, která je definována ve třídě `EntityStats`, je vyvolána při změně maximální hodnoty zdraví, a to v metodě `AddMaxHealthValue`. Slouží například ke správné aktualizaci uživatelského rozhraní. Ve skriptu `HealthBar`, který zobrazuje stav zdraví entit, se na tuto událost přihlašuje metoda `UpdateMaxHealth`. Ta zajišťuje, že se maximální hodnota posuvníku zdraví automaticky přizpůsobí nové hodnotě maximálního zdraví. Zjednodušená reprezentace definice události a jejího využití:

```
public class EntityStats : MonoBehaviour
{
    public event Action OnMaxHealthChanged;

    public void AddMaxHealthValue(float value)
    {
        maxHealth.AddValue(value);
        OnMaxHealthChanged?.Invoke();
        Heal(value);
    }
}
```

Zdrojový kód 2: Definování události

```
public class HealthBar : MonoBehaviour
{
    private EntityStats entityStats;

    private void Start()
    {
        entityStats = GetComponent<EntityStats>();
        entityStats.OnMaxHealthChanged += UpdateMaxHealth;
    }
    private void UpdateMaxHealth()
    {
        slider.maxValue = entityStats.maxHealth.TotalValue();
    }
}
```

Zdrojový kód 3: Přihlášení se k události

2.5 Systém úrovní

Celý systém úrovní je implementován prostřednictvím skriptu `LevelSystem`. Tento skript mimo jiné obsahuje proměnné určující počet zkušenostních bodů potřebných pro dosažení další úrovně, přičemž tento počet se s každou úrovní násobí daným faktorem. Dále zahrnuje proměnné určující procentuální navýšení statistik ostatních nepřátel. Poté obsahuje instanci třídy `LevelDisplay`, která udržuje komponentu textu. Tento text slouží k zobrazení úrovně hráči, který se aktualizuje při každé zvýšení úrovně.

2.6 Systém dovedností

Každá dovednost je reprezentována skriptem `Skill` jako herní objekt, který obsahuje komponenty `Button` a `Image`. Komponenta `Button` umožňuje aktivovat dovednost po kliknutí na její vizuální podobu (`Image`), pokud jsou splněny podmínky pro její zpřístupnění. Mezi tyto podmínky patří například nutnost předchozí aktivace jiné dovednosti, která slouží jako předpoklad pro aktuální dovednost. Skript zároveň zajišťuje logiku samotného odemykání dovednosti. To zahrnuje ověření, zda má hráč dostatečný počet dovednostních bodů, zda daná dovednost již nebyla odemčena, případně zda je možné ji dále vylepšit. V případě úspěšného odemčení se volá metoda `PlayerUnlockSkill`, která zajistí přidání dovednosti do seznamu odemčených dovedností hráče.

Skript hráče pracuje s dovednostmi prostřednictvím dvou metod:

```
1 public bool HasSkill<T>() where T : Skill
2 {
3   return unlockedSkills.Exists(skill => skill is T);
4 }
5
6 public void PlayerUnlockSkill<T>(T skill) where T : Skill
7 {
8   unlockedSkills.Add(skill);
9 }
```

Zdrojový kód 4: Metody hráče pro práci s dovednostmi

Metoda `HasSkill` se poté využívá ke kontrole, zda hráč má odemčenou danou dovednost, aby mohl použít její funkčnost. Funkčnost každé dovednosti určuje metoda `SkillFeature`, která je definovaná ve skriptu `Skill` jako `virtual` (přepisovatelná metoda) a každá dovednost dědí z tohoto skriptu a svou funkci definuje v této přepsané metodě.

Dále skript `Skill` obsahuje instanci třídy `SkillObject`. Tato třída dědí ze třídy `ScriptableObject`, takže využívá výhody popsané v kapitole Skriptovací objekty. Slouží k reprezentaci obsahu dovednosti jako datová struktura. Obsahuje šest atributů: ikonu obrázku dovednosti, jméno, popis, cenu, zda je dovednost odemčena a má možnost vylepšení. Dovednostní objekty lze ve složce projektu vytvořit pomocí kontextového menu (pravé tlačítko myši). Zařazení těchto objektů do nabídky umožňuje atribut třídy `[CreateAssetMenu]`.

2.7 Implementace inventáře

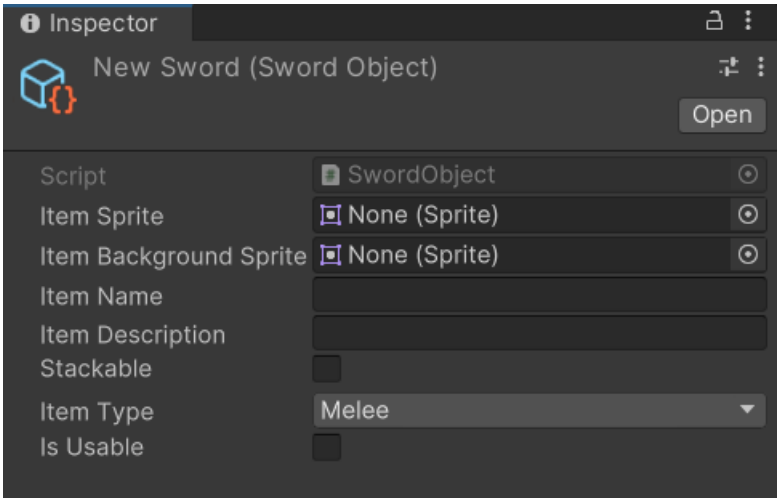
Inventář reprezentuje skript `Inventory`. U inventáře je potřeba jen jedna instance, tudíž je implementovaný jako jedináček, který se v Unity implementuje následovně:

```
1 public class Inventory : Monobehaviour
2 {
3   public static Inventory instance;
4
5   private void Awake()
6   {
7     if (instance == null)
8       instance = this;
9     else
10      Destory(gameObject);
11  }
12 }
```

Zdrojový kód 5: Implementace inventáře jako jedináčka

Implementace jedináčka probíhá ve funkci `Awake`. Pokud instance ještě neexistuje, je aktuální objekt přiřazen jako instance. Pokud instance již existuje, je nově vytvořený objekt zničen, aby se zabránilo vytvoření více instancí.

Předměty inventáře jsou reprezentovány pomocí slovníku `<ItemInstance, InventoryItem>`. Klíč slovníku reprezentuje instanci předmětu, který uchovává jeho statistiky a k dané instanci je přidělený `ItemObject`. `ItemObject` je opět třída dědící ze `ScriptableObject`, která nese potřebné atributy (např. meč viditelný v inspektoru na obr. 6) pro sestavení vlastnosti předmětu. Hodnota slovníku reprezentuje místo slotu v inventáři. Při přidání předmětu do inventáře se přiřadí vlastnostem předmětu jejich slot. Přidání předmětu do inventáře zajišťuje metoda `AddItem`. Ta nejprve zkontroluje, zda je přidávaný předmět mince. Pokud ano, tak se hráči přičte adekvátní počet mincí daný statistikou `price`. Pokud ne, tak se dále kontroluje, zda je možné předmět kumulovat a zda se již nachází v inventáři. V kladném případě se přepočítá cena tohoto předmětu na průměrnou, poté se inkrementuje počet předmětu a nakonec se aktualizuje text s počtem. V záporném případě se pokračuje dále a vytvoří se kopie instance předmětu (nelze zničit sebraný předmět, jelikož při zničení by se zničila i jeho data). Kopii se přiřadí statistiky, najde se volný slot v inventáři (pokud není, objeví se zpráva s informací, že inventář je zaplněn) a tomuto slotu se přiřadí daný předmět a nastaví se jeho grafika.



Obrázek 6: Atributy meče

2.8 Implementace předmětu inventáře

Předmět inventáře je implementovaný skriptem `InventoryItem`, který dědí ze třídy `Monobehaviour` a navíc implementuje tři rozhraní:

- `IDropHandler` – Implementuje metodu `OnDrop`. Ta zpracovává událost přetažení objektu a aktivuje se, když je přetahovaný objekt uvolněn nad cílovým objektem.
- `IPointerEnterHandler` – Implementuje metodu `OnPointerEnter`, která detekuje, zda kurzor vstoupil do oblasti objektu.
- `IPointerExitHandler` – Implementuje metodu `IPointerExit`, která detekuje, zda kurzor již vystoupil z oblasti objektu.
- `IPointerClickHandler` – Implementuje metodu `OnPointerClick`, která zachycuje událost kliknutí kurzoru myši na objekt.

Nutno poznamenat, že uvedené metody jsou volány pouze v případě, že herní objekt, jehož komponentou je daný skript, má aktivní volbu *Raycast Target*. Tato vlastnost komponenty uživatelského rozhraní určuje, zda daný prvek reaguje na události detekované pomocí raycastu – v tomto případě například na kliknutí kurzorem myši.

Skript `InventoryItem` obsahuje kromě obecných atributů, jako jsou například herní objekty s komponentami `Image` (sloužícími k vizuálnímu zobrazení předmětu) a předloha objektu pro zobrazení popisu, také atributy zajišťující logiku práce s předměty. Mezi ty patří například proměnná určující, zda je daný slot aktuálně volný, případně zda obsahuje odkaz na instanci konkrétního předmětu.

Metoda `OnDrop` se zavolá, jakmile se přetáhne předmět do slot a uvolní se tlačítko myši. Při upuštění předmětu se nejdříve zjistí z dat událostí o jaký

přetahovaný předmět se jedná. Zároveň se uloží odkaz na zdroj slotu předmětu inventáře. Následně je logika rozdělena na dvě části – zda se jedná o slot v inventáři v sekci vybavených zbraní nebo se jedná o klasický slot. Pokud se přetahovalo ze sekce vybavitelné zbraně, tak se nejprve zkontroluje, zda se přetahuje správný typ zbraně do tohoto slotu. Pokud ne, tak se předmět vrátí na své původní místo. Pokud ano, tak se zjistí, jestli je daný slot volný a sestaví správné odkazy na instance předmětu a grafiku. Dále je nutné řešit také rodiče objektu ve scéně, jelikož při přetáhnutí se objekt v hierarchii oddělí od aktuálního herního objektu, ke kterému patří. Pokud není daný slot volný, zavolá se metoda `SwapItems` pro záměnu přetahovaného předmětu a předmětu, ze kterého je metoda `OnDrop` volána. Při výměně předmětů se zároveň zkontroluje, zda je tento předmět aktivní zbraní hráče a v kladném případě se nastaví jako nová aktivní zbraň. Jakmile se nejedná o slot vybavitelné zbraně, tak se zkontroluje, zda hráč může přesunout vybavitelnou zbraň do slotu inventáře. Zda je toto hráči umožněno, určuje počet vybavených zbraní. Platí, že pokud má hráč vybavenou zbraň na blízko, tak počet vybavených zbraní musí být menší nebo rovno dvěma, pokud nemá, tak musí být počet menší nebo rovno jedné. Je to proto, aby hráč měl vybavenou alespoň jednu střelnou zbraň. Poté probíhá přesun stejně jako u vybavitelného slotu. Přesun zbraně probíhá obdobně jako v případě běžného vybavitelného slotu, avšak s jednou dodatečnou podmínkou. Pokud hráč přesouvá vybavenou zbraň zpět do inventáře a tato zbraň je zároveň aktivní, je vyvolána metoda, která vyhledá první dostupný index mezi ostatními vybavenými zbraněmi. Následně je tento index použit k nastavení příslušného parametru v řídicím systému animací a odpovídající zbraň je nastavena jako nová aktivní.

Metody reagující na vstup a výstup kurzoru z oblasti objektu pracují s aktivováním herních objektů. Jedná se buď o zobrazení tlačítka, které se objeví při najetí kurzorem na aplikovatelný předmět, nebo o zobrazení dialogového okna. Dialogové okno se aktivuje v případě, že má předmět v inventáři přiřazenou instanci. Při najetí kurzoru na tuto instanci probíhá konfigurace dialogového okna ve třech krocích. Nejprve se jeho obsah nastaví pomocí metody `SetupDialogueWindow`:

```
1  private void SetupDialogueWindow()
2  {
3      StringBuilder sb = new();
4      ItemObject itemObj = itemInstance.itemObject;
5
6      sb.AppendLine($"<size=32><b>{itemObject.itemName}</b></size>"
           );
7      sb.AppendLine($"{itemObject.itemDescription}\n");
8
9      foreach (Stats stat in itemInstance.stat)
10     {
11         int value = (int)stat.TotalValue();
12         sb.AppendLine($"{stat.type.ToString()}: {value.ToString()
               }");
13     }
14
15     descriptionGO.GetComponentInChildren<TextMeshProUGUI>().text
           = sb.ToString();
16 }
```

Zdrojový kód 6: Sestavení obsahu dialogového okna.

Pro práci s řetězci se využívá třída `StringBuilder`. Nejprve se uloží odkaz na objekt předmětu a poté se do proměnné `sb` uloží dva řádky – název, který je tučným a větším písmem (komponenta *TextMeshProUGUI* podporuje HTML tagy), poté popis předmětu a následně se projdou všechny názvy typů statistik a jejich hodnoty. Nakonec se komponentě textu nastaví její obsah na hodnotu `sb` do finálního řetězce. Poté se nastaví herní objekt dialogového okna na aktivní a v poslední fázi se spustí koprogram, který nastavuje pozici okna. Nastavení pozice okna musí probíhat přes koprogram namísto klasické metody, jelikož se musí spočítat velikost okna, kterou okno zabírá, což by bez prodlevy nebylo možné. Koprogram vrací null – `yield return null`, což znamená, že sestavení pozice dialogového okna je odložena na další snímek.

Systém událostí sám o sobě neobsahuje metodu implementující dvojklik, proto je tato logika řešena metodou `OnPointerClick`. Uvedená metoda nejprve ověřuje, zda bylo provedeno kliknutí levým tlačítkem myši, dále zda je povoleno prodávání předmětů (v oblasti obchodů) a jestli se nejedná o vybavenou zbraň. Pokud jsou všechny tyto tři podmínky splněny, uloží se čas kliknutí. Pokud se do definované doby realizuje další kliknutí (což realizuje dvojklik), tak se daný předmět, na který byl proveden dvojklik, prodá.

2.9 Ekonomika hry

Celá ekonomika hry je řízena skriptem `Coins`, který je opět vytvořen jako jedináček. Obsahuje privátní proměnnou, která udržuje počet dostupných mincí. K ní jsou příslušné metody pro přidání minci, odebrání mincí či nastavení mincí. Společně s voláním těchto metod se vždy volá i metoda `UpdateCoinText`, která nastavuje správné zobrazení počtu mincí v uživatelském rozhraní.

2.10 Rozvržení předmětů

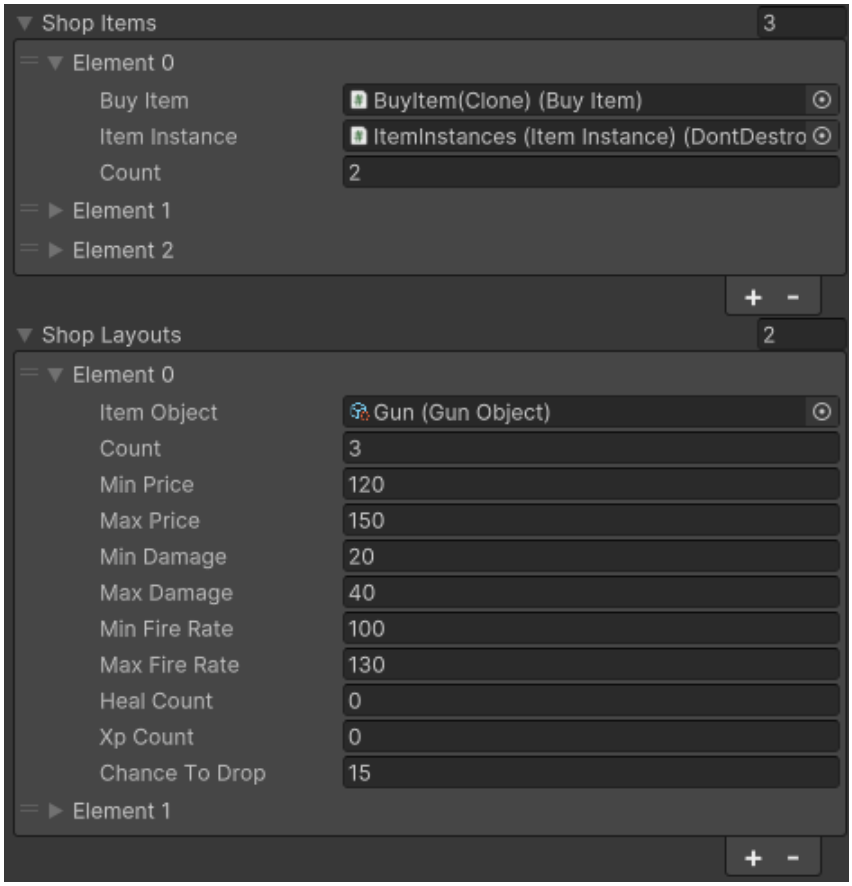
Jaké předměty hráč získává, buď v obchodech, nebo z nepřátel určuje třída `ItemLayout`. Tato třída nedědí ze žádné třídy a je serializovatelná. Poskytuje rozvržení předmětu, který je dán odkazem na objekt předmětu. Třída obsahuje kromě hodnot, které mají nastavit předmětům (např. minimální a maximální cena, poškození, rychlost střelby), také hodnoty jako je šance na výběr konkrétního předmětu, nebo počet, kolikrát se má předmět objevit. Dále ke třídě přísluší tři statické metody – jedna pro výběr předmětu, který hráčovi padne, kde se sečte pravděpodobnost všech předmětů a pak se náhodně jeden vybere na základě vypočtené pravděpodobnosti. Zároveň se přitom využije metoda pro získání dostupných předmětů. V té se projdou všechny předměty, na které má hráč nárok (využije se například metoda, zda má hráč nárok na padnutí zbraně, viz Metody hráče pro práci s dovednostmi). Poslední metoda třídy zvyšuje hodnoty statistik předmětů. Tato metoda se využívá především při zvýšení úrovně hráče, kdy dochází k navýšení hodnot statistik jednotlivých předmětů.

2.11 Implementace obchodu

Každý obchod má komponentu skriptu s názvem `Shop`. Tento skript má 2 serializovatelné herní objekty – první herní objekt s uživatelským rozhraním obchodu a druhý herní objekt je předloha objektu, která reprezentuje kupovaný předmět. Kupovaný předmět se skládá z dalších herních objektů – obrázek pozadí, obrázek samotného předmětu, textu, který udává cenu předmětu a obrázek mince a nakonec tlačítko, které realizuje nákup. Dále skript obsahuje logickou proměnnou `canBuy`, která signalizuje, zda hráč může nakupovat v obchodě. Tato proměnná je nastavována v metodách `OnTriggerEnter2D` a `OnTriggerExit2D`, což jsou metody ze třídy Monobahaviour, které řeší kolizi. Pokud se tedy hráč dostane do oblasti obchodu (tzn. začne kolidovat s obchodem), tak se v první metodě nastaví proměnná na pravdu. Pokud hráč obchod opustí, je pomocí druhé metody nastavená proměnná na nepravdu. Poté se v `Update` metodě kontroluje vstup uživatele pomocí podmínky zda stiskl klávesu `B`. Pokud tedy stiskne tuto klávesu a proměnná `canBuy` je nastavena na pravdu, otevře se nabídka obchodu – herní objekt uživatelského rozhraní se nastaví na aktivní.

To, jaké předměty bude obchod nabízet, udává seznam rozvržení předmětů. Ten je veřejný, takže se vždy v inspektoru nastaví ty hodnoty, jaké budou dané předměty mít (viz ukázka na obr. 7). Předměty, které obchod nabízí, se uchovávají v seznamu objektů třídy `ShopItem`. Tato třída je serializovatelná a obsahuje pouze tři vlastnosti – instanci kupovaného předmětu, instanci předmětu a počet, kolikrát lze předmět zakoupit. Kupovaný předmět reprezentuje třída, která obsahuje a nastavuje veškerou grafiku jednotlivých předmětů v obchodě včetně dialogového okna či tlačítka. K tlačítku je přiřazena událost, která při kliknutí spustí metodu pro zakoupení konkrétního předmětu předaného jako argument.

Sestavení předmětů obchodu provádí metoda `SetupShop`. Ta je volána buď v metodě `Start` každého obchodu, nebo při otevření nabídky obchodu, pokud si



Obrázek 7: Obchod z inspektoru

hráč odemkl nový typ zbraně. Tato metoda vytvoří nové instance předmětů, kterým dá vlastnosti na základě rozvržení předmětů a přiřadí jim statistiky. Nakonec každou instanci předmětu přidá do obchodu pomocí metody `AddItemToShop`. Tato metoda vytvoří novou instanci předmětu obchodu, kde ji přiřadí zmiňované tři vlastnosti. Počet je náhodně určený, odkaz na instanci předmětu má k dispozici, pouze si vytvoří instanci kupovaného předmětu. To obnáší instanciaci herního objektu z předlohy kupovaného předmětu. Nově vytvořenému hernímu objektu přidá komponentu kupovaného předmětu a nastaví ho předmětu obchodu. Nakonec nastaví hodnotu ceny předmětu.

Koupení předmětu z obchodu realizuje metoda `Buy`. Ta má jeden argument – kupovaný předmět. Nejdříve se nalezne předmět v seznamu předmětů obchodu a zjistí se jeho cena. Pokud se daný předmět nalezl a hráč nemá dostatek peněz, objeví se notifikace informující o nedostatečném obnosu peněz. Jinak je možné koupení realizovat tak, že se sníží peněžní hodnota předmětu, poté se zjistí, zda lze zakoupit předmět vícekrát. Pokud ano, sníží se počet, pokud ne, odstraní se předmět z obchodu. Následně se odečtou hráči peníze a získá malé množství zkušenostních bodů.

2.12 Vypadnutí předmětů

Vypadávání předmětů z nepřátel je zajištěno skriptem `DropController`. Ten musí být ve scéně pouze jeden, tudíž je opět implementován jako jedináček. Obsahuje tři atributy – předlohu objektu pro vypadlý předmět, poté herní objekt, který je rodičem vypadlých předmětů a seznam rozložení objektů. Dále obsahuje dvě metody — veřejnou, která se volá při vypadnutí předmětu a vybírá jeho rozložení, a neveřejnou, která na základě tohoto rozložení předmět sestaví. Ta nejprve provede instanciaci objektu z předlohy, poté jí přidá komponentu instance předmětu, nastaví objekt předmětu, ze kterého společně s rozložením sestaví konečný předmět se statistikami. Nakonec se nastaví správný obrázek předmětu, jeho pozice a jemného odražení při vypadnutí z nepřítele.

2.13 Kontrolní body

V úrovních jsou kontrolní body spravovány skriptem `CheckpointController`. Ten obsahuje metodu dosažení kontrolního bodu, která nastaví herní objekt předchozího kontrolního bodu na neaktivní a nový kontrolní bod nastaví na bod z parametru metody. Tuto metodu vždy volá kontrolní bod, do kterého hráč dorazil. Další metoda vrací pozici posledního kontrolního bodu, kterou využívá hráč, když dojde k jeho oživení. Poslední metoda slouží k dodatečnému nastavení při nastavení pozice během oživení. Provádí samotné oživení hráče a zároveň zajišťuje oživení nepřátel. Ožijí pouze nepřátelé, které hráč dříve porazil a kteří se nacházejí na pozicích od posledního kontrolního bodu. Zároveň oživení nepřátel zařizuje nastavení původních pozic nepřátel, nastavení opět maximálního zdraví, aktivuje objekt jak nepřátele, tak jeho ukazatele zdraví a nakonec přepne do nečinného animačního stavu.

Při kolizi s hráčem, zachycené metodou `OnTriggerEnter2D` (přístupnou z Unity ze třídy `MonoBehaviour`), skript zavolá metodu `ReachCheckpoint` z kontroleru kontrolních bodů a předá jí v parametru `checkpoint` odkaz na sebe.

2.14 Portál

Portál je prostředek pro objevení a zjevení se ze scény. Je to herní objekt, který má komponentu Animator a BoxCollider2D. Obsahuje dvě logické proměnné, kde první z nich určuje, zda se portál nachází ve scéně tutoriálu a druhá, zda se jedná o startovní portál. Pokud se jedná o startovní portál, tak ve `Start` metodě se nastaví hráč na neaktivního. Když poté portál bude v kolizi s hráčem, tak se zjistí, zda je aktivní scéna tutoriálem (to je pro případ, aby se zničily herní objekty, které se normálně neničí při přechodu mezi scénami) a poté se nastaví v animaci proměnná na zavření portálu a nakonec se hráč nastaví na neaktivního. Pokud se portál uzavře, je zavolána metoda ze spouštěče událostí animace pro načtení další úrovně. To probíhá tak, že se zjistí index aktuální scény a počet celkových scén ve hře, z těchto proměnných se zjistí, zda je následující

scéna další úroveň. Pokud ano, načte se scéna s číslem o jednu vyšší než aktuální, pokud ne, načte se scéna hlavního menu.

2.15 Audio

Veškeré audio ve hře je řízeno skriptem `AudioManager`. Každý zdroj audia si skript uchovává v proměnných typu `AudioSource` (případně jejich polích). Pro většinu těchto zvukových zdrojů jsou k dispozici metody, které přehrají odpovídající audio. Odlišné přehrání má například metoda pro spuštění hudby při souboji s bossem. Nejprve se zkontroluje, jestli právě hraje hudba na pozadí. Pokud ano, zastaví se koprogram, který přehrávání zajišťuje. Samotné přehrávání hudby na pozadí funguje pomocí koprogramu, který běží v nekonečné smyčce. V každém cyklu se nejprve zjistí, zda již nějaká hudba hraje -– pokud ano, zastaví ji. Poté náhodně vybere novou skladbu, tu spustí a následně čeká, dokud nedohraje. Poté se celý proces opakuje. Čekání je řešeno příkazem `yield return new WaitForSecondsRealtime(currentBGMusic.clip.length)`, který zajistí pozastavení na dobu trvání skladby v reálném čase. Díky tomu se cyklus nezpozdí ani v případě, že je například otevřený inventář nebo jiné herní menu, které pozastavuje běžný herní čas.

2.16 Hlavní menu

Hlavní menu při spuštění hry ovládá skript `MainMenuController`, který je jedináčkem. Má celkem čtyři tlačítka – startovní a pokračující tlačítka, tlačítko pro tutoriál a ukončující tlačítko. Z toho první tři jsou serializovatelná. Dále má čtyři proměnné typu řetězec, které nesou názvy scén. Tyto názvy jsou důležité, neboť reprezentují scény, které mají načítat. Podle toho zda má hráč rozehranou hru či dokončený tutoriál se tyto tlačítka buď zobrazí nebo ne. Při spuštění má menu vzhled jako na obr. 8.

2.17 Menu ve hře

Menu ve hře nabízí tři možnosti – pokračovat ve hře, nastavení a návrat do hlavního menu. V nastavení se nachází dva posuvníky. První je pro nastavování hlasitosti hudby pozadí a druhý pro nastavování hudby zvukových efektů. Ovládání hlasitostí je realizováno pomocí *AudioMixer*, což je nástroj v Unity, který umožňuje seskupovat zvukové stopy dohromady a poté s nimi pracovat jako s celkem. Při posunutí posuvníku se zavolá metoda pro nastavení hlasitosti dané skupiny dle parametru charakterizující danou skupinu. Nastavování hlasitosti je možné vidět na obr. 9.

Obrázek 8: Výchozí menu

Obrázek 9: Nastavení zvuku

3 Popis samotné hry

Tato kapitola se zaměřuje na podrobný popis samotné hry z hlediska jejího průběhu, herních mechanik a základních principů, na nichž je postavena.

3.1 Předměty

Ve hře se nachází několik druhů předmětů. Mezi nejvýznamnější patří zbraně, které lze rozdělit do čtyř kategorií:

- pistole,
- kuše,
- brokovnice,
- meče.

Jednotlivé typy zbraní se odlišují svými charakteristikami. Pistole a kuše představují univerzální střelné zbraně s delším dosahem. Kuše se vyznačuje vyšší kadencí střelby, avšak nižším způsobeným poškozením. Brokovnice naopak dosahuje výrazně vyššího poškození na úkor nižší kadence a kratšího dosahu. Meč představuje zbraň určenou pro boj zblízka. Všechny tyto předměty lze vybavit.

Dalším typem předmětu jsou předměty aplikovatelné. Mezi ně patří především lektvary, které lze rozdělit do dvou skupin – lektvary zdraví a lektvary zkušeností. Lektvar zdraví může hráč využít v případě, že nemá plné zdraví. Po použití lektvaru zkušeností se hráči doplní určitý počet zkušenostních bodů.

3.2 Inventář

Hráč spravuje své předměty prostřednictvím inventáře. Inventář obsahuje celkem 21 slotů a dále 4 speciální sloty pro vybavení zbraní (viz obr. 10). V místech pro vybavené zbraně je každé ze čtyř míst určeno pro konkrétní kategorii zbraní. Pro vybavení zbraně je nutné ji přetáhnout z inventáře do příslušného slotu. Platí přitom pravidlo, že hráč musí mít vybavenou alespoň jednu střelnou zbraň.

Pokud má hráč vybaveno více typů zbraní, může mezi nimi přepínat pomocí kláves: `1` pro pistoli, `2` pro kuši a `3` pro brokovnici.

Jsou-li v inventáři dostupné aplikovatelné předměty, přejetím myši na předmět se zobrazí tlačítko „USE“. Pokud jsou splněny podmínky pro použití, lze daný předmět aplikovat stisknutím tohoto tlačítka. V případě, že chce hráč aplikovat daný předmět vícekrát, při stisknutí tlačítka souběžně přidrží klávesu `Shift`.

Obrázek 10: Inventář

3.3 Zkušenosti

Během průběhu hry hráč postupně získává zkušenostní body, a to zejména za vyvíjení aktivity, eliminace nepřátel či realizaci nákupu v obchodě. Po dosažení určitého počtu zkušenostních bodů dochází ke zvýšení úrovně postavy. Požadovaný počet zkušeností pro další úroveň se s každým postupem zvyšuje.

Zvýšením úrovně se zlepší bojové statistiky postavy a zároveň hráč získá jeden dovednostní bod. V případě, že se jedná o úroveň, jejíž číslo je násobkem pěti, získá hráč dva dovednostní body.

3.4 Dovednosti

Důležitým herním prvkem jsou dovednosti. Dovednosti odemykají nové herní mechaniky nebo vylepšují ty stávající. Každá dovednost má stanovenou cenu pro odemknutí či vylepšení.

Po otevření stromu dovedností, jak je patrné z obr. 11, je v jeho horní části zobrazen aktuální počet dostupných dovednostních bodů. U jednotlivých dovedností je dále uvedena jejich cena. Při najetí kurzorem myši na konkrétní dovednost se zobrazí její daná funkcionalita. Má-li hráč dostatečný počet dovednostních bodů a zvolí danou dovednost, dojde k jejímu odemknutí.

Ve hře je k dispozici celkem osm dovedností:

- **Kuše** – odemyká kuši jako novou herní zbraň,
- **Brokovnice** – odemyká brokovnici jako novou herní zbraň,
- **Dvojitý skok** – umožňuje provést druhý skok ve vzduchu,

- **Úskok (dash)** – umožňuje krátkodobý úskok do strany,
- **Zvýšení poškození zbraní** – zvyšuje poškození způsobené pistolí, kuší a brokovnicí,
- **Zvýšení maximálního zdraví** – zvyšuje maximální hodnotu zdraví.

Obrázek 11: Strom dovedností

3.5 Obchod

V průběhu hry hráč narazí na několik obchodů, ve kterých je možné zakoupit herní předměty. Rozhraní obchodu, které je zachyceno na obr. 12, se otevře stisknutím klávesy B. Každý předmět je určen svou cenou. Při najetí kurzorem myši na konkrétní předmět se zobrazí jeho popis, případně doplňující informace, jako jsou statistiky. Po aktivaci tlačítka „BUY“ je vybraný předmět automaticky přidán do hráčova inventáře.

Obchod nabízí pouze ty předměty, které může hráč aktuálně používat. Například pokud dosud neodemkl dovednost umožňující používání kuše, tento typ zbraně se v nabídce obchodu nezobrazí. Kumulovatelné předměty lze zakoupit opakovaně.

Obrázek 12: Uživatelské rozhraní obchodu

3.6 Nepřátelé

Ve hře se nachází sedm druhů nepřátel (na obr. 13):

- **Skřet** – základní typ nepřítele, v ničem nevyniká,
- **Létající oko** – vyznačuje se vysokou rychlosti pohybu,
- **Houba** – vyznačuje se rychlým útokem,
- **Kyklop** – jeden z nejsilnějších nepřátel, má speciální útok kouzlem,
- **Noční zrození** - vyznačuje se velmi silným útokem,
- **Kostlivec** - jednou za čas může na obranu využít svůj štít
- **Lukostřelec** - jediný nepřítel na dálku, střílí šípy.

Obrázek 13: Nepřátelé

3.7 Tutoriál

Hra nabízí možnost tutoriálu, kde se hráč seznámí se základními mechanikami ve hře. Postup tutoriálem je doprovázen dialogovými okny, které seznamují hráče s mechanikami a jejich využitím. Během tutoriálu si hráč vyzkouší všechny dříve zmíněné mechaniky včetně soubojů se všemi nepřáteli.

3.8 Průběh hry

Při spuštění hry je hráč vybaven pouze jednou zbraní – pistolí. Pokud bude hráč v průběhu hry eliminován (a to buď nepřítelem nebo pádem do propasti), tak ztratí náhodný počet předmětů v intervalu od nuly do poloviny celkového počtu předmětů. Zároveň ztratí polovinu svých peněz. V průběhu hry se hráč snaží postupně získávat zkušenosti, eliminovat nepřátele, odemykat nové dovednosti, vybavovat se lepšími zbraněmi, procházet všemi úrovněmi, až do poslední úrovně, kde pokud porazí finálního bosse, vyhrává.

Závěr

Cílem práce bylo vytvoření 2D RPG hry v Unity. Hra nese název „Shadow of the Oaks“ vycházející z prostředí, do kterého je zasazena. Ve hře byly implementovány základní herní systémy typické pro RPG hry – inventář s předměty, bojový systém, obchodování, zkušenosti a dovednosti. Při vývoji byl kladen důraz na to, aby bylo možné hru dále rozšiřovat o nové funkcionality. Mezi potenciální rozšíření patří například přidání dalších animačních stavů (jako je střelba ve vzduchu, lezení po žebříku či sklouznutí po stěně), rozšíření o nové dovednosti nebo přidání dalších typů nepřátel.

Během vývoje jsem prohloubil své znalosti v oblasti vývoje počítačových her, získal hlubší porozumění architektuře herních systémů, principům objektově orientovaného návrhu v Unity a práci s jeho komponentovým modelem. Za největší výzvu při vývoji hry považuji nalezení vhodné grafiky, a to jak pro uživatelské rozhraní, tak pro animace hráče i nepřátel, která je nedílnou součásti procesu vývoje každé hry.

Conclusions

The aim of this project was to create a 2D RPG game in Unity. The game is titled "Shadow of the Oaks" reflecting the world in which it is set. Core RPG systems were implemented — an item-based inventory, a combat system, trading, experience and skill progression. During development, emphasis was placed on ensuring the game can be extended with new features. Potential future enhancements include adding further animation states (such as shooting while airborne, climbing ladders, or wall-sliding), expanding the skill set, or introducing additional enemy types.

Throughout development, I deepened my knowledge of computer game creation, gained a deeper understanding of game-system architecture, the principles of object-oriented design in Unity, and working with its component model. The greatest challenge I encountered was sourcing appropriate graphics — for both the user interface and the animations of player characters and enemies, which is an integral part of the development process for any game.

A Obsah elektronických dat

`bin/`
: Adresář se spustitelnou verzí hry vytvořenou v Unity.

`doc/`
: Adresář s textem bakalářské práce ve formátu PDF, včetně všech souborů potřebných k jeho sestavení.

`src/`
: Adresář s projektem Unity, obsahující všechny zdrojové soubory potřebné pro otevření ve vývojovém prostředí.

`README.txt`
: Textový soubor s návodem na spuštění hry a projektu v Unity, včetně postupu.

Literatura

[1] Wikipedia. *Unity (game engine)*. [online] [cit. 2023-10-05]. Dostupný z: ⟨https://cs.wikipedia.org/wiki/Unity_(hern%C3%AD_engine)⟩.

[2] Technologies, Unity. *Introduction to components*. [online] [cit. 2025-05-01]. Dostupný z: ⟨https://docs.unity3d.com/Manual/Components.html⟩.

[3] Technologies, Unity. *Assets and media*. [online] [cit. 2025-05-01]. Dostupný z: ⟨https://docs.unity3d.com/Manual/assets-and-media.html⟩.

[4] Technologies, Unity. *Programming in Unity*. [online] [cit. 2025-05-01]. Dostupný z: ⟨https://docs.unity3d.com/6000.2/Documentation/Manual/%09%20scripting.html⟩.

[5] Technologies, Unity. *Order of execution for event functions*. [online] [cit. 2025-05-01]. Dostupný z: ⟨https://docs.unity3d.com/6000.1/Documentation/Manual/execution-order.html⟩.

[6] Technologies, Unity. *Prefabs*. [online] [cit. 2025-05-01]. Dostupný z: ⟨https://docs.unity3d.com/Manual/Prefabs.html⟩.

[7] Technologies, Unity. *ScriptableObject*. [online] [cit. 2025-05-01]. Dostupný z: ⟨https://docs.unity3d.com/Manual/class-ScriptableObject.html⟩.

[8] Technologies, Unity. *Script serialization*. [online] [cit. 2025-05-01]. Dostupný z: ⟨https://docs.unity3d.com/6000.1/Documentation/Manual/script-serialization.html⟩.

[9] Technologies, Unity. *Splitting tasks across frames*. [online] [cit. 2025-05-01]. Dostupný z: ⟨https://docs.unity3d.com/6000.1/Documentation/Manual/coroutines.html⟩.

[10] Technologies, Unity. *Introduction to scenes*. [online] [cit. 2025-05-01]. Dostupný z: ⟨https://docs.unity3d.com/Manual/CreatingScenes.html⟩.

[11] roninMo. *2d-Intro-Unity-Project*. [online] [cit. 2025-05-01]. Dostupný z: ⟨https://github.com/roninMo/2d-Intro-Unity-Project/tree/main⟩.

[12] Technologies, Unity. *Rigidbody2D*. [online] [cit. 2025-05-01]. Dostupný z: ⟨https://docs.unity3d.com/ScriptReference/Rigidbody2D.html⟩.

[13] Technologies, Unity. *Animator*. [online] [cit. 2025-05-01]. Dostupný z: ⟨https://docs.unity3d.com/ScriptReference/Animator.html⟩.

[14] Mixkit. *Free Game Sound Effects*. [online] [cit. 2025-05-01]. Dostupný z: ⟨https://mixkit.co/free-sound-effects/game/⟩.

[15] Freesound. *Freesound*. [online] [cit. 2025-05-01]. Dostupný z: ⟨https://freesound.org/⟩.

[16] Pixabay. *Free Sound Effects*. [online] [cit. 2025-05-01]. Dostupný z: ⟨https://pixabay.com/sound-effects/⟩.

[17] Uppbeat. *Free sound effects for video editing*. [online] [cit. 2025-05-01]. Dostupný z: ⟨https://uppbeat.io/sfx⟩.